EPM/RT-95/01

CASE-tool Learnability in a Software Engineering Course

Germinal Boloix
Pierre N. Robillard

gratuit

Département de génie électrique
et génie informatique

École Polytechnique de Montréal
Janvier 1995

Dépôt légal, novembre 1993
Bibliothèque nationale du Québec
Bibliothèque nationale du Canada

Pour se procurer une copie de ce document, s'adresser:

Les Éditions de l'École Polytechnique
École Polytechnique de Montréal
Case postale 6079, succ. Centre-ville
Montréal, (Québec) H3C 3A7
Téléphone: (514) 340-4473
Télécopie: (514) 340-3734

CASE-tool Learnability in a Software Engineering Course

Germinal Boloix, Pierre N. Robillard
December 1994
Ecole Polytechnique de Montréal
Département de Génie Electrique et de Génie Informatique
C.P. 6079 succ Centre-ville
Montréal, Québec H3C 3A7
Tel. (514) 340-4031 - Fax. (514) 340-4063
boloix@rgl.polymtl.ca
pnr@rgl.polymtl.ca

Abstract

CASE-tool learnability is analyzed in a lab session setting associated with an undergraduate software engineering course. The organization of the lab is presented, together with the impact of CASE-tool learnability on software development exercises. CASE-tool learnability affects the breadth and depth of students' exercises because of the time limitations of the lab sessions. CASE-tool complexity is determined by the number of functions that have to be exercised by the students versus those available in the tool. Students are considered novice users of tools, and their learning time is considered an external indicator of CASE-tool complexity.

Keywords: CASE-tool learnability, CASE-tool complexity, Software Engineering course, Laboratory sessions

Type of paper: experience report

1.0 Introduction

There are severe time limitations to mastering the functionality of CASE tools in university courses. The time allocated for a course can easily be exceeded, depending on the breadth and depth of the expected number of CASE tool functions to be learned. CASE tool complexity, determined by the number of functions available in a tool, is only one aspect affecting student performance. There are other, background aspects as well, such as familiarity with the problem domain, knowledge about software representation methods or techniques, knowledge about programming languages, and familiarity with the operating system and hardware platforms.

Using CASE tools in software engineering courses has been prescribed in many computer science curricula [Z94], [H94], [GW92], [LL92], [W88]. In most courses, theory sessions and project assignments which involve CASE tool usage are given together, with the students producing a prototype of a software system at the end of the course. There is widespread recognition that the learning curve is quite steep when both aspects are taught during one course. It has been suggested that instead of only one course, two courses in sequence are required, one to give theory concepts and introduce students to the CASE tools, the other project-oriented to develop a software system.

In this paper, a teaching experience in the second type of setting (i.e., first course: theory and CASE tools, second course: project development) is presented. Only the impact of CASE tool complexity during the first course of this sequence is being analyzed. In our curriculum the second course is not compulsory, therefore simulation of software development activities is required to prepare the students. The experience during the last two semesters is presented.

There are serious time limitations in laboratory sessions. Students are expected to complete an assignment during the time allocated to a session (in our case 3 hours). In practice, much more time is required to learn some of the functions of the tools and produce a result. Students find it hard to be productive within the allocated time. It is well known in industry that learning a tool to capacity takes several months [K92].

Complexity is evaluated from the point of view of the user of the CASE tool, in this case the student. CASE tool learnability is a subject that has not received much attention from the research community. There are methods for evaluating and comparing human computer interface development tools [HS91], and evaluating text editors [RM83]. SEI has proposed a guide for classifying and assessing software engineering tools [WGPMF87]. These methods basically evaluate functionality and usability of similar tools. There are approaches to evaluating CASE tools by establishing benchmarks to develop and modify a system [ComputerWorld90]. See the appendix for a brief description of these approaches.

In the software engineering lab, we are faced with the problem of analyzing the complexity of different tools, each of which addresses a different stage of the software life cycle. Determining the complexity of CASE tools requires analysis of the user interface using the tool and the knowledge necessary to master the usage of the tool. To be proficient with a tool, the user needs to understand the syntax and semantics of the underlying method (i.e., diagrams, icons, relationships, and so on). The complexity of the user interface is evaluated according to the number of functions to be exercised and the time required to learn them. Students are considered novice users.

In the context of the software engineering lab, each CASE tool is addressed separately following the stages of software development. Issues regarding integration of these tools is mentioned, but not stressed, during the course. Each tool is evaluated separately to determine how complex it is. Most of the tools are very complex, target functions have to be preselected (i.e., those functions of interest for the lab session). This activity is performed by the instructor before each session, when the description of the excercise is being prepared.

This paper is organized as follows: Section 2 gives some background on the evolution of the software engineering course and its lab component. Section 3 presents the organization of the lab. Section 4 presents a description of the system developed during the lab sessions. Section 5 presents an analysis of the complexity of the CASE-tools used in the lab. Section 6 presents some conclusions and areas for research.

2.0 Software engineering course

Courses in software engineering are normally given using what we call the 'theory-project' approach for one semester. This type of setting requires extra work by the students because of the need to learn theory and practice concepts at the same time, and, most of the time, practice concepts by their own. There is a tendency to split the 'theory-project' approach into a two-courses-in-sequence approach which involves a *theory-tool* course and a *project* course [RMD94]. Instructors experience in teaching in both types of settings for different institutions provide a wider understanding of the acceptable theory and practice load to teach in the same course. The experience reported in this paper corresponds to the last two semesters of 1994, specifically regarding lab sessions (theory-tool course).

Theory-Tool Course

The software engineering course is part of the undergraduate program in computer engineering. It follows a sequence of topics using a standard book [S93]. Lab sessions are organized in the same order as the theory sessions: estimation, planning, architectural design, detail design, coding and analysis of quality. Theory sessions are syncronized with lab sessions such that theory is given before the practical session. Last semester there were 60 students in the course, divided into two sections. Within each section students worked in groups of two.

The objective of the lab is twofold, simulating software development stages for a system (the same system description is used for all the sessions) and becoming familiar with CASE tools on-line, and accessing tutorials and tool documentation. There are secondary objectives, like gaining experience working in a group, becoming exposed to state-of-the-art workstations and understanding the choice of alternatives during design and implementation.

The initial stage before a lab session concerns preparation. The process involves identifying the important aspects to understand during the corresponding stage of the software life cycle, the evaluation of the specific CASE tool available during the session (i.e., trying the CASE tool) and preparation of the written description of the exercise.

Three-hour practical sessions are scheduled every two weeks, covering the above-mentioned stages of the software life cycle. Each practical session refers to the same editorial system description (see below). For each practical session, an outline is prepared. The outline is divided into four parts: first, the objective of the session; second, a list of aspects for reflection during this stage of the life cycle; third, the expected work to be done; and, finally, the list of deliverables to be submitted.

Project Course

This course follows the theory-tool course, and is not a compulsory part of the curriculum. It is a practical course which follows a common software development process. Students, organized into groups, have to develop a system according to the specifications provided. The experience gained with the tools during the first course in the sequence is invaluable to the students. Support from user representatives is given by an instructor or an expert in the application domain. Technical support is given by personnel involved in CASE-tool installation. Experience with this course is reported in [RMD94].

3.0 Organization of the Lab

Lab sessions for the software engineering course are organized according to the CASE tool used. Tools are presented in sequence, following software development stages. Case tools used in the lab sessions are organized as upper-CASE tools: estimation, planning and architectural design, and lower-CASE tools: detail design, coding and static analysis.

Tool's functionality

A summary of each tool functionality is as follows:

COSTAR: This is a cost estimation tool based on the Constructive Cost Model (COCOMO) described by Boehm [B81]. Estimates of project duration, staffing levels and costs are produced with this tool. Trade-offs and 'what-if' analysis can be performed to generate the final optimal project plan.

Microsoft Project: This is a tool for planning project activities. Functions to create a project, manage a project and produce reports are integrated within this tool. Different types of project control diagrams are generated, such as PERT charts and Gantt charts. Resource assignment and several types of reports are generated with this tool.

STP: This is a tool for analysis and design. Several editors are available within STP: dataflow editor, data structure editor, structure charts editor, entity relationship editor, state transition editor, as well as other editors used in software development. Dictionary facilities are available to keep the documentation of software systems.

Schemacode: This is a tool for editing algorithms using stepwise refinement concepts. A top down approach to break down major procedures into smaller, less abstract sub-procedures, allows programmers to organize their programming tasks. Refinements, and constructs are used to develop algorithms. Refinements are abstractions like chapters in a book, whereas constructs can be sequential, conditional or repetitive, following the concepts of current programming languages.

SUIT: This is a user interface toolkit using C language. SUIT permits the creation of prototypes either empty or running under a program written in C; the location of user interface elements can be modified interactively without changing the source code.

DATRIX: This is a static analyzer from source code. It allows evaluation of source code quality by computing several software metrics. Software components can be identified for further analysis when measured values are compared with user-defined standards. Coherence and consistency of source code can be verified using this tool.

Tool evaluation

An evaluation was performed to obtain a high level understanding of the tools. SEI's software tool evaluation approach was used partially [WGPMF87]. The evaluation parameters were the following:

- Ease of use: One measure of a tool's effectiveness is the ease with which the user can interact with it.
- Power: The extend to which the tool 'understands' the objects it is manipulating and the extend to which simple commands can cause major effects. Power is also demonstrated by reasonable performance achieved through efficient use of the computing resource.
- Robustness: The reliability of the tool, the performance of the tool under failure conditions, the criticality of the consequences of tool failures, the consistency of tool operations, and the way in which a tool is integrated into the environment.
- Functionality: The accuracy and efficiency with which the tool supports the underlying methodology affects the understandability and performance of the tool, as well as determining the quality and usefulness of tool outputs.

A subjective scale of one to five was used to evaluate the tools. A one indicating the lowest evaluation and a five representing the highest evaluation. The following table shows a summary of the evaluation.

Table 1: CASE-tool evaluation

	Ease of use	Power	Robustness	Functionality
COSTAR	3	4	4	5
Project	4	4	4	5
STP	4	4	4	5
Schemacode	4	4	4	5
SUIT	3	4	3	4
DATRIX	4	4	3	5

According to Table 1, the evaluation shows that the tools are fairly similar. Exceptions include SUIT which requires C programming language, COSTAR that is a text-based interface and DATRIX that is a tool developed for research instead of commercial purposes.

Lab philosophy

The general philosophy behind the laboratory course is not to lead students by the hand, but rather to encourage them to develop their own ability to explore an unfamiliar computing environment. The main resources required in the lab course are the following:

- Computer: two different labs are required, one using PCs, and the other using Sun workstations. Both require printing facilities.
- Operating system: one lab using DOS/Windows, and the other using Unix and Xwindows.
- CASE-tool installation: Each tool has to be available before the sessions.

Tool support (e.g., installation of tools, UNIX tune-up environment): PC-based tools have the software installed for general use, without additional effort from the students. Unix-based tools have the software installed, but students have to include specific tool parameters in defining their Unix environment.

The description of a system to be used throughout the sessions is defined at the beginning of the semester. The same system description is used throughout the semester. Written descriptions of the assignments for each lab are given, with an indication of the work to be done. Before starting each session, half-hour explanation of what is requested and how to do it is given in a classroom. Examples are given, showing what students are expected to hand in, and the steps to follow using the tools to generate deliverables. An example also provides a view of the kind of work students are expected to produce. The suggested steps are as follows:

- Define the exercises to be performed by the students during each session
- Check out the tool to verify correct installation
- Solve the problem before the session to determine the time it takes

Each session lasts 3 hours. Normally, students require additional time. They have to determine the availability of workstations. The steps during each session are to:

- Explain the objectives of the session, what is expected and how to proceed
- Give support to the students during the session

The activities of the instructor after the session are to:

- Keep a log of the problems occuring during the session
- Correct the students' results

A brief summary of the tools used during the sessions is as follows:

Estimation: The COSTAR tutorial requires about 40 minutes to complete. There were 16 pages of documentation available. The user manual has 120 pages.

Planning: A full Microsoft Project tutorial requires over seven hours to complete. The user manual contains 660 pages.

Architecture: The Software Through Pictures (STP) basic tutorial requires about 2 hours to complete. There are 47 pages of documentation in this tutorial. The number of pages of documentation in the user manual was 770.

Detail design: The Schemacode tutorial required about 30 minutes to complete. There are 18 pages documenting the tutorial. The number of pages in the user manual is 54.

Implementation: C language using SUIT (Simple User Interface Toolkit). The SUIT tutorial requires about 30 minutes to complete. There are 16 pages of documentation for the tutorial. The user manual is composed of 162 pages.

Static analysis: DATRIX. The tutorial requires about 40 minutes to complete. There are 33 pages documenting the tutorial. The user guide is composed of 164 pages.

The computing environment requires that the students be familiar with the hardware platforms and the operating systems. A great deal of energy is expended to teach the students to use the tools and carry out the development stages, rather than simply learning system basics. There were cases of students becoming familiar with the technology during the course, and this definitely affected their performance. Technology for the tools is as follows:

- COSTAR and Microsoft Project run in DOS/Windows environments on a PC.
- The other tools run in Unix, Xwindows environments on Sun workstations.

4.0 System example: Editorial

The philosophy of the lab sessions has evolved over the years. At the beginning, for each session, a different system example was introduced. The disadvantage of this was that it did not simulate a real software development project. Lately, during the last two semesters of 1994 reported in this paper, a single system description has been used throughout all the lab sessions. The advantage of providing a single system description is that students become familiar with the details of the system incrementally, while simulating the development of a system. The standard description is called the Editorial system.

Using the same problem system during the lab sessions has the advantage of preparing the students for a real software development experience. As the next course (i.e., only project) is not mandatory, the students get a practical feel for problems that occur during a real project. Combining the same exercise with theory classes is an excellent way of integrating theory and practice concepts.

The Editorial system is taken from a structured analysis book [Y89] and adapted to our needs. Editorial organizational background is given in the description. A summary of the estimated system volumes is also given: published books, actual and potential authors, number of orders per day, annually, and so on. Characteristics of billing and payments and the notion of returns are also mentioned. These are common notions in the publishing business.

The description contains a list of 40 system events, a list of 8 main functions and a list of 10 files. The estimated size of the system is 10.000 lines of C code. This indicates that it is a large problem to be solved by students, even if small from an industry perspective.

The lab sessions are mostly oriented towards the tools, leaving the simulation of the life-cycle stage as a secondary objective. However, students are free to interview the instructor, who acts in this case as the user representative, to gain an understanding of details of the system. Details of each lab session are as follows,

The estimation session required decomposing the system into components, and a determination of the number of lines of code for each component to produce an estimate. Students need some advice on how to translate functional specifications into software size. LOC is the main parameter on COCOMO; other parameters are related to personnel, project, software and computer characteristics; cost factors are also required for the estimate. Once these parameters are input, the tool provides several reports on cost and schedule information. A report explaining the students' findings is requested.

The planning session required the subdivision of project activities into phases and tasks. Duration of tasks and allocation of resources completes the assignment. Gantt and network charts are generated using the tool. A report summarizing the findings is produced.

The architectural design session demanded a high-level design of the system using structured techniques.Together with the exercise description, a detailed specification of one function was supplied, to provide students with the logic behind the functions. Data flow diagrams, including the context diagram for the full system and the first and second levels of decomposition for one of the functions, were requested. Data structure diagrams for the files were also produced. Documentation produced with the tool was produced, but a report of students' findings was not expected.

The detail design session demanded the realization of algorithms for one of the functions in the system. A detailed specification was provided for this function. Formats of the user interface and printed reports were expected from the students. Updated data structures were also requested. Documentation produced with the tool and graphical forms were produced, but a report with findings was not requested.

The implementation session required the development of a running prototype following the specifications of the detail design phase. Instructor access to the students' directory for purposes of executing the program was needed. Documentation consisted of a program listing and images of user interface screens and examples of printed reports. No special report on findings was requested.

The static analysis session demanded obtaining the metric analysis of the previously developed software. Documentation was requested in the form of tool printouts and a report of students' findings.

5.0 Analysis of CASE tool complexity

Faced with the complexity of the tools, the instructor had to identify, for each tool, a subset of important functions for the lab session, such that the students would be able to complete the exercise in a reasonable amount of time. An evaluation of CASE-tool functionality is presented in the following diagram. The outside circle represents the capacity CASE tool functionality, and the inside circle represents the subset of functions to be experienced by the students.

FIGURE 1. Functionality of the tools

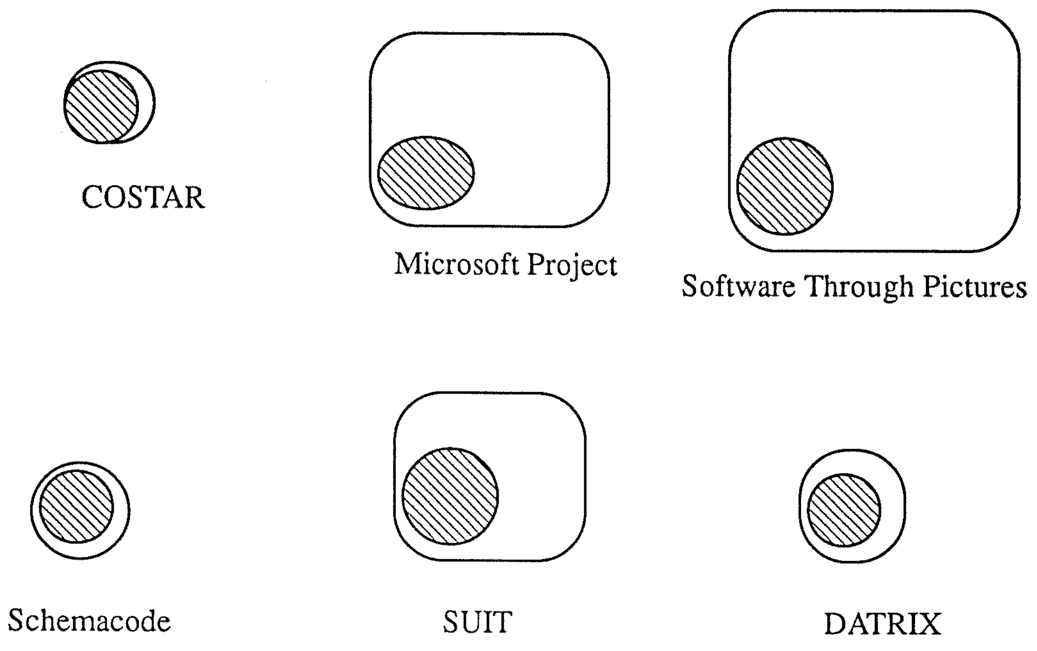

Figure 1 shows that STP was the more complex tool from a user perspective, followed by Microsoft Project and SUIT. COSTAR was the simplest tool, folowed by Schemacode and DATRIX. In general, the amount of functionality to be experienced by students was equivalent for all the tools. However, the amount of time required to complete the lab session was also related to other parameters, such as the method or the problem domain and the degree of detail of the work to be done.

Background knowledge on the subject matter was given during theory sessions. Students were not provided with additional theoretical background during lab sessions. Students were given instructions on what had to be done (i.e., expected results) and how to use the tool (i.e., steps required to set up the tool and to solve the problem). Simulation of a stage of software development required some preparation by the students, for example reviewing the description of the Editorial system and the description of the specific session. Tool information included setting up the environment (e.g., Unix parameters) and identifying important tool functions and the order of steps required to complete the exercise.

Estimation tool:

COSTAR did not require identification of tool functions before the lab session, even though there were some functions not used to solve the problem (e.g., establishing function points to estimate LOC). The tutorial for this tool required about 40 minutes to complete. A short time was needed to complete the exercise, and an average of 5 hours was sufficient to produce the results, which included a report of their findings. Figure 1 shows that the estimated functionality experienced by students with this tool was almost the full functionality of the tool.

Planning tool:

Because of the large number of tool functions available in Microsoft Project, a previous selection of functions was needed to complete the exercise for the planning stage. Instead of 6 hours to complete the tutorial, only 1.5 hours were required following the identified tool functions. Once having completed the tutorial, the exercise of preparing a software development plan for the Editorial started. It took the students an average of 6 hours to complete the exercise, which included a report of their findings. Figure 1 shows that the estimated functionality experienced by students was only 20% of the full functionality of the tool.

Architectural design tool:

The basic Software Through Pictures tutorial takes two hours to complete. It still required one hour to complete the tutorial when the tool functions to use during the exercise were selected: Data Flow Editor and Data Structure Editor. Examples of diagrams were presented to the students. The exercise required an average of 7 hours to complete. There was no report to hand in, only the printed output from the tool. Figure 1 shows that the functionality experienced by students was less than 20% of the full functionality of the tool.

Detail design tool:

Schemacode did not require identification of tool functions before the lab session, however examples of pseudocode listings were shown to the students. The tool's tutorial requires about 30 minutes to complete. The exercise required an average of 6 hours to complete. No final report was requested, only pseudocode listings and layouts of screens and reports. Figure 1 shows that the functionality experienced by the students was close to the full functionality of the tool.

Implementation tool:

There was no identification of call-back functions for SUIT, even though it would be convenient to identify some functions in future courses. The tutorial takes about 30 minutes to complete. An example of a program was presented to the students before the session. The exercise required an average of 16 hours to complete. No report was requested, only the program listing and printouts of screens and reports. Figure 1 shows that the functionality experienced by students with the tool was 30% or less than the capacity of the tool.

Static analyzer tool:

There was no identification of tool functions for DATRIX. The tutorial requires about 40 minutes to complete. The exercise demanded an average of 4 hours to complete. A report was requested, as well as the printouts produced with the tool and an analysis of students' findings. Figure 1 shows that the functionality experienced by students was around 50% of the tool's functionality.

Table 2 shows a summary of results from an elapsed-time perspective. The first column indicates the estimated time to complete the tutorial, as suggested in vendor documentation or estimated by the instructor where this information was not available. See the appendix for a detailed account of the time it takes to complete the tutorial for one of the tools. The second column shows the time reported by the students to learn each tool for the subset of required functions. DATRIX and COSTAR required the least amount of time to learn, followed by STP and Schemacode; SUIT required the greatest amount of time to learn (this is a tool that requires browsing C functions from its 162 pages of documentation). The third column presents the actual time taken to complete the exercise, which included the allocated 3-hour lab session. DATRIX required the least amount of time, followed by COSTAR, Schemacode, Microsoft Project and STP. SUIT required the largest amount of time (an average of 300 executable lines of code were produced by the students).

Table 2: Average time taken to perform Lab activities

	Time to complete tutorial (hours)	Time to learn tool (hours)	Time to complete exercise (hours)
COSTAR	.6	1.5	5
Microsoft Project	1.5	3	6
Software Through Pictures	1	1.8	7
Schemacode	.5	1.7	6
SUIT	.5	4	16
Datrix	.6	1	4

There are several components to complexity: problem domain complexity (i.e., knowledge about the system to be developed), method or technique complexity (i.e., knowledge about a phase of the software life cycle and its representation techniques) and external tool complexity (i.e., knowledge about the functionality of the tool). Internal tool complexity is not addressed in this study.

The first two columns of Table 2 are directly related to CASE-tool complexity, whereas the third colum is related to a mixture of problem, technique and tool complexities. As we are concentrating on CASE-tool learnability rather than problem or technique complexity, an effort to identify sources of tool complexity was performed. Let us analyze these results for each tool:

COSTAR: This is a cost estimation tool. The interface is text-based and very simple. The underlying model is much more complex. Most of the functions of this tool were accessed: determining delivered source instructions, cost driver estimation, decomposition into components, establishing cost per staff-month, and generating the various reports. More than 80% of the functions were exercised.

Microsoft Project: This is a tool for planning project activities. The functions required to solve the exercise were primarily related to planning a project: establishment of activities, their sequence, time to perform the activities and resources required to perform the activities. This represents 20% of the available functions of the tool. Functions related to controlling the project were not addressed.

STP: This is a tool for analysis and design. The interface is relatively simple. Only a few STP functions were required to solve the problem. DFE was used to draw the data flow diagrams and DSE was used to draw the data structure diagrams. This represents less than 20% of the available major functions of the tool. In addition, each of these functions was used only partially.

Schemacode: This is a tool for editing algorithms. The interface is relatively simple. It supports the creation of sequential, repetitive and conditional constructs using the concept of stepwise refinement. Source code statements are input during editing, and automatic generation of source code can be performed if source code statements are provided. More than 80% of the functionality of this tool was exercised by the students.

SUIT: This is a user interface toolkit using C language. The location of user interface elements can be modified interactively without changing the source code. To solve the problem, students had to use several SUIT call-back functions. About 30% of the available functions had to be exercised.

DATRIX: This is a tool for static analysis from source code. The external complexity of this tool, its user interface, is relatively simple, whereas the internal complexity of the tool is much greater. From the students' perspective, handling the tool is straightforward. Only a subset of functions was required to perform the analysis of their programs. Most of the functions of the tool were accessed: specify information on a project (a project is composed of several programs), analyze sample source files, display analysis results on the screen, print reports of analysis results. About 50% of the functions were exercized.

Computer professionals are continually faced with new hardware or software systems. Students need to become confident in their own abilty to read manuals, use help systems, and get informal assistance from peers to become productive. An analysis of CASE tool documentation was performed. See the appendix for an analysis of pages of documentation for one of the tools. Results on the number of pages per tool are presented in Table 2.

Table 3: Pages of documentation per tool

	Tutorial (pages)	Documentation Required (pages)	User Manual (pages)
COSTAR	37	37	(120) + 37
Microsoft Project	~50 on-line	~50 on-line	660 + ~210 on-line
Software Through Pictures	47	37 + 47	723 + 47
Schemacode	18	18	36 + 18
SUIT	16	~60 + 16	162 + 16
Datrix	33	60 + 33	131 + 33

The first column in Table 3 presents the number of pages describing the tutorial, which is provided by the vendor of the tool. The '~50 on-line' means there are approximately the equivalent of 50 pages of documentation on-line, instead of printed. The second column shows the documentation available to solve the proposed problem. This is the documentation provided with the lab manual. For some tools the tutorial contained enough documentation to solve the problem (e.g., COSTAR, Schemacode). In other cases additional documentation was necessary (e.g., STP, SUIT). In one case on-line documentation was supplied (Microsoft Project) and in other cases definitions of metrics were supplied in additional documentation (DATRIX was provided with 60 pages of metrics definitions '+' 33 pages of tutorial). The '~60 + 16' means approximately 60 pages of the user manual were required, in addition to ('+') 16 pages of the basic tutorial, to complete the exercise. The last column presents the number of pages in the user manual in addition to ('+') the number of pages in the basic tutorial. '(120) + 37' means that according to COSTAR documents there are 120 pages in the user manual, which was not available for our analysis.

Table 3 is a way of establishing the functionality of the tools according to pages of documentation. It is possible to compare these results with those in Figure 1, which shows that STP is the more complex tool, followed by Microsoft Project and SUIT. COSTAR, Schemacode and DATRIX are less complex. See the appendix for an anlysis of the number of pages of the user manual for one of the tools.

6.0 Conclusions

As software engineering courses evolve towards the introduction of CASE tools, problems of tool complexity limit the breadth and depth of tool functions to learn during a semester. The time available to learn the tools during a course tends to be insufficient. Steps to distribute the learning process over several courses is required. We have presented the two-courses-in-sequence approach (i.e., theory-tool, project), specifically the theory-tool experience. Other approaches may involve the introduction of tools over the full career path of students.

There are several components to complexity from the point of view of students: problem domain complexity, method or technique complexity and tool complexity. Only tool complexity has been addressed in this study, and further research is required to analyze the impact of different sources of complexity on the performance of students. For a software engineering course it is fundamental to select the subset of functions to be used during a lab session such that the students are able to complete their assignments in a reasonable amount of time. Simulating the development of one system throughout the semester facilitates the accomplishment of the exercises.

CASE-tool learning time increases when some functions are not documented. This requires finding help in the lab to clarify the steps to perform an action. Also, if the tool does not provide feedback when performing some functions,, it is difficult to guess what has happened.

The amount of tool knowledge experienced by students during the course is limited. Additional work would be necessary by the students to master the more complex tools. A few months of continuous usage are necessary to master a tool. However, with this hands-on course, the students are in a better position to select the set of tools to be used in the next course in the sequence (project course) or in a particular software development environment.

Research on tool evaluation up to now has been mostly about similar types of tools. A common framework for comparing dissimilar tools constitutes an interesting subject for research. The ability to compare the complexity of different tools used to develop systems may help to facilitate their adoption. Tools vary not only in their functionality or stage of application in the development life cycle, but also on their interface characteristics (e.g., text-oriented or window-oriented) and their internal complexity.

Tool integration is an area that has received some contribution from the research community, although for the purposes of a software engineering course there is a lack of integration in the commercial tools analyzed. The output of each tool can be used as input to the next tool in the sequence. CASE tools in the course are studied one by one during each lab session. By using the same system example, students are able to identify what information from one phase is being used in the next phases. However, the automation of tool integration requires tool compatibility, which was only available in a few cases during the course (e.g., Schemacode could be used to develop the code in C with SUIT, and the C code can be analyzed with DATRIX).

The issue of whether these are the right tools to introduce in a software engineering course is a subject of research. Only a subset of possible tools required during software development are used in the course. The introduction of additional tools in the same course would be complicated. A possibility would be to exchange some tools (e.g., instead of the estimation tool, introduce a testing tool) depending on the direction of the course, whether oriented towards technical or managerial aspects.

References

[B81] Boehm, B. 'Software Engineering Economics', Prentice-Hall, Inc. 1981.

[ComputerWorld90] 'Product Review', Computer World, twelve issues focusing on integration of CASE tools and 4GL languages. June 1990 to September 1991.

[GW92] Grau, J.K.; Wilde, N. 'Use of the Individual Exchange Project Model in an Undergraduate Software Engineering Laboratory', LNCS 640, Software Engineering Education, Proceedings of the SEI Conference, San Diego, California, USA, October 1992, C. Sledge (ed), Springer-Verlag, Berlin, Heidelberg, 1992.

[H94] Horton, T.B. 'Using Commercial CASE Environments to Teach Software Design', LNCS 750, Software Engineering Education, Proceedings of the 7th. SEI CSEE Conference, San Antonio, Texas, USA, January 1994, Jorge L. Diaz-Herrera (ed), Springer-Verlag, Berlin, Heidelberg, 1994.

[HS91] Hix, D.; Schuman, R.S. 'Human-Computer Interface Development Tools: A Methodology for their Evaluation', Communications of the ACM, Vol. 34, No. 3, March 1991, pp.74-87.

[K92] Kemerer, C.F. 'How the Learning Curve Affects CASE Tool Adoption', IEEE Software, May 1992.

[LL92] Lively, W.M.; Lease, M. 'Undergraduate Software Engineering Laboratory at Texas A&M University', LNCS 640, Software Engineering Education, Proceedings of the SEI Conference, San Diego, California, USA, October 1992, C. Sledge (ed), Springer-Verlag, Berlin, Heidelberg, 1992.

[RM83] Roberts, T.L.; Moran, T.P. 'The Evaluation of Text Editors: Methodology and Empirical Results', Communications of the ACM, Vol. 26, No. 4, April 1983, pp.265-283.

[RMD94] Robillard, P.N.; Mayrand, J.; Drouin, JN. 'Process Self-Assessment in an Educational Context', LNCS 750, Software Engineering Education, Proceedings of the 7th. SEI CSEE Conference, San Antonio, Texas, USA, January 1994, Jorge L. Diaz-Herrera (ed), Springer-Verlag, Berlin, Heidelberg, 1994.

[S93] Schach, S.R. Software Engineering, second edition, Richard D. Irwin, Inc., and Aksen Associates, Inc., 1993.

[W88] Werth, L.H. 'Software Tools at the University: Why, What and How', LNCS 327, SEI Conference 1988, Fairfax, Virginia, USA, April 1988. Gary A. Ford (ed), Springer-Verlag, Berlin, Heidelberg, 1988.

[WGPMF87] Wood, W.G.; Gold,L.R.; Pethia, R.; Mosley, V.; Firth, R. 'A Guide to the Classification and Assessment of Software Engineering Tools', Software Engineering Institute, CMU/SEI-87-TR-10, August 1987.

[Y89] Yourdon, E. 'Modern Structured Analysis', Prentice-Hall, 1989.

[Z94] Zalewski, J. 'Cohesive Use of Commercial Tools in a Classroom', LNCS 750, Software Engineering Education, Jorge L. Diaz-Herrera (ed), Proceedings of the 7th. SEI CSEE Conference, San Antonio, Texas, USA, January 1994, Springer-Verlag Berlin Heidelberg, 1994.

Appendix: Approaches to evaluating software tools

Software Tool Evaluation [WGPMF87]

Ease of use: One measure of a tool's effectiveness is the ease with which the user can interact with it.

Power: The extent to which the tool 'understands' the objects it is manipulating and the extent to which simple commands can cause major effects. Power is also demonstrated by reasonable performance achieved through efficient use of the computing resource.

Robustness: The reliability of the tool, the performance of the tool under failure conditions, the criticality of the consequences of tool failures, the consistency of tool operations, and the way in which a tool is integrated into the environment.

Functionality: The accuracy and efficiency with which the tool supports the underlying methodology affects the understandability and performance of the tool, as well as determining the quality and usefulness of tool outputs.

Ease of insertion: The ease with which a tool can be incorporated into the target organization or environment.

Quality of Support: The level of training required and provided, the cost of maintenance agreements, the understandability of its documentation.

Text Editor Evaluation [RM83]

Time to perform basic editing tasks by experts: The time it takes expert users to accomplish routine text modifications is measured by observing expert users as they perform a set of benchmark tasks.

Cost of errors for experts: The effect of errors in an editor is measured by the error time, which is the time cost of errors on the benchmark tasks. The course of a typical error includes committing the error, discovering it, correcting it, and then resuming productive behavior.

Learning basic editing tasks by novices: The ease of learning to use an editor is tested by actually teaching novice subjects, individually, to perform editing taks.

Functionality over a wide range of editing tasks: The range of functionality available in an editor is measured by analyzing the editor against a checklist of tasks covering the full task taxonomy.

Human-Computer Interface Development Tool Evaluation [HS91]

Functionality: Indicates what the tool can do; that is, what interface styles, techniques and features can be produced by the tool for a target application interface.

Usability: Indicates how well the tool performs its possible functions, in terms of ease-of-use (how easy or difficult the tool is to use) and human performance (how efficiently the tool can be used to perform tasks).

Specification/Implementation Techniques: For specifying and implementing a user interface, ranging from programming paradigms to graphical metaphors.

CASE-tool Benchmark Evaluation [ComputerWorld90]

CASE tools are observed in action over a three-day period during which the vendor team solves the case study project costing system, an application that is familiar to most information systems professionals. The team mission is to demonstrate the capability of the CASE tool to deliver complete and complex business solutions under 'live fire' conditions.

A rating of Poor, Fair, Good, Very good and Excellent was assigned to each category.

Level of completion: Assessment of the number of functions implemented.

Speed of development: Assessment of the total effort required to implement the case study.

Speed of maintenance: Assessment of the effort required to perform changes on the functionality of the system.

Documentation: Assessment of the quality and amount of documentation produced by the tool.

Integration of tools: Assessment of the level of integration among CASE tools.

End-user language: Assessment on the availability of end-user languages which consult the database.

Appendix: duration of a tutorial

Microsoft Project:

1. Learning the basics:

- Using Microsoft windows, 15 minutes
- What is project management, 5 minutes
- What is Microsoft Project, 20 minutes
- Tour of the tool bar, 5 minutes
- Working with views, 25 minutes

2. Creating a project:

- Entering the tasks, 35 minutes
- Outlining tasks, 30 minutes
- Assigning task relationships, 30 minutes
- Entering resources, 25 minutes
- Creating calendars, 40 minutes
- Analyzing the plan, 35 minutes

3. Managing a project:

- Emphasizing information, 30 minutes
- Tracking progress, 25 minutes
- Managing resources, 25 minutes
- Adjusting schedules, 15 minutes

4. Producing reports:

- Changing the look, 20 minutes

- Printing views and reports, 25 minutes

TOTAL = 405 minutes.

Appendix: number of pages of user manual

Software Through Pictures:

- Introduction to Software Through Pictures, 13 pages

- Getting started, 16 pages

- Creating the project directory structure, 13 pages

- Basic tutorial, 51 pages

- Data flow diagram editor (DFE), 68 pages

- Data structure editor (DSE), 51 pages,

- Entity relationship editor (ERE), 62 pages

- Structure chart editor (SCE), 53 pages

- Control flow editor (CFE), 9 pages

- Control specification editor (CSE), 57 pages

- State transition editor (STE), 14 pages

- Picture editor, 35 pages

- Object annotation editor (OAE), 15 pages

- Data dictionary, 15 pages

- Printing diagrams, 29 pages

- Document preparation system (DPS), 46 pages

- Using STP: advanced tutorial, 50 pages

- Transition diagram editor (TDE), 30 pages

- Rapid to use prototype development system, 74 pages

- Generating 2167A reports, 69 pages

Total = 770 pages

[redacted]